2004:052 HIP

BACHELOR'S THESIS

Distribution of Incremental Disk Images

A systems management environment implemented in Rembo Toolkit

Johan Hallbäck

Luleå University of Technology

BSc Programmes in Engineering
Department of Computer Science and Electrical Engineering
Division of Computer Science

2004:052 HIP - ISSN: 1404-5494 - ISRN: LTU-HIP-EX--04/052--SE

Distribution of Incremental Disk Images

A systems management environment implemented in Rembo Toolkit

by

Johan Hallbäck

johan.hallback@ltu.se

Department of Computer Science and Electrical Engineering
Luleå University of Technology
S-971 87 Luleå, Sweden

November 2004

Examiner
Andreas Nilsson,
Luleå University of Technology, Sweden

Published 2004
Printed in Sweden by University Printing Office, Luleå

Abstract

This thesis deals with scalable cloning of modern operating system installations in a network, heterogeneous in terms of hardware and software requirements. The main problem examined is distribution of pre-installed software for Microsoft Windows (incremental images), using an implementation written during this thesis work in the pre-OS management platform Rembo Toolkit. In addition, the problems of computer group configuration managment and pre-boot operating system customization will be treated. The purpose is not only to give an example of a management tool suitable in a certain network environment, but also to address some customization issues arising when cloned operating systems and incremental images interact with widely used network services. However, some drawbacks in the incremental imaging approach derived from certain applications rather than Rembo Toolkit has been identified, thus limiting its overall usability.

Acknowledgements

This work has been carried out at the Computer Support Group in the Department of Computer Science and Electrical Engineering (CSEE), Luleå University of Technology, Sweden. It is my final project of the Bachelor of Computer Science programme.

I would first of all like to thank Per Björn (supervisor), Andreas Nilsson (examiner), Daniel Larsson, Mattias Stahre and Krister Engberg of the Computer Support Group for giving me the opportunity to work with them, both during this project and afterwards as a member of their group.

I wish to give extra credits to Helena Sandström and Johan Karlsson at the Division of Computer Science and Networking. Helena for being both carrot and stick when the goal seemed unreachable and Johan for excellent LaTeX support when needed. Also, thanks to Annika Sällström at CDT for lending me a portable computer during the completion of this thesis.

Last but not least I would like to acknowledge my friends who made me do something besides work during the last year: Lunch mates Pål Nilsson and Niklas Lundström, B&D (Bastu & Dinner team), Henrik Elmunger, Frederik Schmid and fellow musician Stefan Granlund for providing coffee blacker than darkness. And of course, thanks to Elisabeth and my family and relatives in southern Sweden for always supporting me in my studies.

Luleå, November 2004

Johan Hallbäck

Contents

Man ska alltid overkilla allt.
– Robert Stjärnström

Always overkill everything.
– Robert Stjärnström

Chapter 1

Introduction

The thesis begins with a description of the problem field and outlines the remainder of the thesis.

In many large scale system administration environments network-based cloning[1] of hard drive content is of high importance. This is a common situation in systems with many users and a set of identical, public computers, where users expect their software environment to be the same regardless of where they are sitting. Not only does cloning provide convenience to the user, cloning tools also ease the task of system administrators as maintenance is ideally limited to the reference computer.

Many software tools exist to aid the system administrator in creating and deploying hard drive content to a set of computers. The tools differ a lot in terms of features, efficiency/scalability, ease of use and the ability to customize the deployment. But with more advanced operating systems and the increasing use of different network services, cloning is likely to require post-installation treatment in order to keep certain parts of the hard drive content unique. This issue is treated as one of the big problems with cloning. However, many contemporary cloning tools are able to manipulate the operating system settings on a per machine basis.

One of the undoubtedly most competent solutions in this field is the Rembo Toolkit (abbreviated Rembo throughout this text), developed by the Swiss company Rembo Technology SaRL. Put briefly, Rembo is a small, network-booted graphical operating environment for x86 PC:s, including a programming/scripting language. This language, called Rembo-C, is syntactically a mix between C, Java and HTML and contains an API with a huge amount of functions to allow for customized disk imaging and operating system configuration. Rembo contains support for creating self-contained base images of several operating systems, including Linux and all current versions of Microsoft Windows. It

[1] Cloning in this context is the process of duplicating installed operating systems and other software from a reference computer and restoring the data on several other computers. The intermediate file is called an *image*.

also supports *incremental imaging*, i.e. the process of comparing two partitions or images and store the differences in a third image.

The work presented in this thesis has been conducted at the Computer Support Group at the Department of Computer Science and Electrical Engineering (CSEE) at Luleå University of Technology, Sweden. The Computer Support Group (abbreviated CSG in this text) is responsible for about 1200 computers, and with the majority being PC workstations containing different flavors of Windows and/or Linux, Rembo was purchased with the vision to solve most of the PC installation issues. Thus, the goal of this work has been to design and implement a Rembo-based solution customized to simplify the task of the CSG, a process which should detect any problems associated with disk imaging and solve them with Rembo if reasonably possible. In particular, the suitability if using incremental imaging for distributing installed Windows software has been examined. This work resulted in a cloning wizard for Rembo called "SM RemboAdmin".

The report is organized as follows. In Chapter 2, the task and the problems resulting in the need for this work is introduced in greater detail, together with a more comprehensive overview of Rembo. Previous solutions and possible alternatives will also be presented. In Chapter 3 some common customization problems associated with cloning of different operating systems will be discussed, along with proposed solutions. An evaluation of incremental imaging with Rembo will also be presented. Chapter 4 describes the custom-built computer grouping solution this implementation has yielded in, along with examples of how this can be configured. The last section will conclude the work and deal with present problems and further work.

Appendix A contains explanations of technical abbreviations in the text. The thesis ends with a user's manual and a maintenance guide for SM RemboAdmin in appendices B and C, respectively.

Chapter 2

Background

In this chapter a more detailed description of the tasks and current deployment problems of the Computer Support Group is given. A thorough presentation of the Rembo concepts and features is also included to show what problems an implementation of the product is able to solve. In addition, a quick overview of possible alternatives to Rembo is presented along with a brief history of cloning at the department, wrapped up by a formulation of the project goal that resulted in this thesis.

2.1 The problems at hand

2.1.1 The need for cloning

The CSG comprises of four full time employed system administrators with a maintenance responsibility of about 1200 computers. The vast majority of computers are standard PC:s, either used as workstations by the staff (researchers, teachers and administrative personnel), or as anonymous desktops in four different computer labs used by students taking courses at the departments. Among many other things, it is the task of the CSG to make sure these computers are usable, i.e. that operating systems and relevant software tools are installed, configured and updated. Due to the amount of computers it is impossible to do all software installation and configuration manually, which is why network-based cloning tools with various extents of competence have been used since the first PC lab was introduced at the department back in 1996. Ideally, all configuration should be included in the reference image before further distribution.

So, the need for cloning is imminent, but the concept itself doesn't alone solve the problems anymore. The reason is the ever increasing complexity of the operating systems; the Windows NT family[1] in particular. Modern applications

[1]The loose term *Windows NT family* contains the OS:es Windows NT/2000/XP/2003 since they all are derived from the NT architecture. The term Windows 9x will also be used throughout the thesis when speaking generally about Windows 95/98/ME.

and network services tend to need more and more customization after cloning, as configuration data often is tied to a specific computer. This makes flexibility a firm requirement for the cloning tools. In fact, Microsoft has in [7] stated that cloning Windows is only supported if it is done before the GUI part of the installation is reached, which is before any host customization at all has taken place. Microsoft has also released software called Sysprep to undo the customization part of a client in order to provide supported cloning. This tool will be presented in Section 2.3.3.

2.1.2 Differences in software and hardware

All PC:s differ vastly in terms of operating systems and applications installed, and the choice for each workstation depends on the end user. The amount of disciplines at the departments range from media technology, computer science and electrical engineering to mathematics, with all these disciplines using their own software tools. Due to this, there is no possibility of having a single installation suiting all users as the total number of applications is too large.

Differences in hardware configuration is also an issue. Inevitably, all computers are not identical in terms of devices (e.g. network interfaces, sound- and video cards) and other peripherals (lab instruments, video cameras etc), which causes problems with missing or wrong device drivers when cloning without care. To help solve this the computer labs are roughly divided into categories by discipline, where each lab is supposed to have a relevant set of applications installed. Furthermore, the computers in these labs have identical hardware to facilitate cloning. With hardware and software grouped together, the next task was to find a cloning tool which supports both Linux and different flavors of Windows, and this is where Rembo Toolkit came at hand.

2.2 An introduction to Rembo Toolkit

2.2.1 Concept overview

The server part of Rembo consists of a *Preboot eXecution Environment* (PXE) boot server, supplying a small network-based operating environment to a set of client computers. The BIOS of a client PC uses its PXE Boot ROM compatible network interface as boot device in order to download the operating environment from the Rembo server. In Chapter 4 the booting procedure will be examined in detail to see how booting and grouping the clients are done by using the *Dynamic Host Configuration Protocol* (DHCP).

In many aspects, the downloaded program is like a small operating system in the shape of a boot loader. Featured with a high resolution, mouse controlled windowed graphical user interface and advanced disk imaging operations the client side of Rembo acts as tool for installation and configuration between the BIOS and the "real" operating system. The server side is responsible of storing and transferring disk images and other files on behalf of the hosted clients. However, it is far more complex than a common file server when it comes to network protocols and efficiency.

The strengths of the Rembo operating environment lies in its scripting language Rembo-C, a C/Java/HTML hybrid accompanied by an enormous API of

function calls. The client has its own byte code format for executing server-located precompiled Rembo-C plugins as well as interpreting scripts on the fly. Categories of operation the API supplies range from file management (both locally and on server), network operations, GUI manipulation, hardware detection and system calls. In addition, mechanisms for group management and user authentication exist to protect the most critical operations. This also allows for full customization as each set of computers (possibly originating from the same server) can download their their own Rembo implementations.

2.2.2 Features

This section summarizes the key features of Rembo taken into consideration when purchasing the product. It should provide hints of the problems an implementation can help solve.

- *Advanced imaging.* Rembo uses a *structure-based cloning* approach, meaning that it is always aware of the filesystem[2] on the reference computer. Thus, it treats the contents of a disk image as files and directories instead of raw data[3]. With this information Rembo is able to merge images and partitions, possibly of different types. Furthermore, differential snapshots between partitions and images can be taken to create *incremental images*. With this feature it is possible to distribute images of already installed Windows applications at boot time.

- *File system transparency.* As with Unix, everything in the Rembo operating environment is files, abstracted by the underlying file system. There exists internal file system support for local disks, opened browsable disk images and Windows NT registries[4], GUI objects etc. Besides making programming easier, this provides for manipulating files and registries both on local disk and on server located images.

- *Operating system configurability.* The aforementioned filesystem transparency allows for full customization of system and application patches. Besides unlimited possibility to write custom patches, the Rembo-C API includes a set of already implemented registry patches to simplify the most common configuration tasks with the Windows NT family. The API also contains a set of string manipulation functions. This allows for file patching[5], useful for pre-boot personalizing computer settings stored in text files.

- *Efficiency.* The client and server share a multicast protocol for efficient deployment of multiple workstations concurrently. To further relieve the

[2] As of Rembo Toolkit 2.0, the supported file flavorssystems are FAT (all flavors, MS-DOS/Windows 9x), NTFS (Windows NT/2000/XP), EXT2/3 (Linux) as well as the usual floppy and CD-ROM formats.

[3] This second approach is called *sector-based cloning*. When used it copies an entire partition or disk to a file, possibly ignoring any unallocated space on the source medium.

[4] The Windows registry is comprised of a couple of binary files representing a hierarchical, file system-alike systemwide repository of configuration parameters and their values. It is intended as a way for programmers to integrate applications with the system software. A good overview can be found in [4].

[5] *File patching* in this context is the process of inserting a snippet of text in a file at positions indicated by some particular pattern.

network, all files transferred with this protocol are cached locally on each client in the unpartitioned space after the last partition. This makes subsequent restorations mere local file copy operations only limited in speed by the local disk. In addition, both server and client maintain a shared repository[6] containing files present in two or more images, providing great scalability in terms of storage. It also limits the amount of data to be transferred during cloning and restoration.

- *Usability.* Custom made operations can often be made more usable by providing a simple graphical user interface. In Rembo, this consists of adding strings of HTML/Javascript-alike code to different GUI elements created with the API functions.

2.3 Past and present competitors

2.3.1 BpBatch

BpBatch is a free remote boot processor, developed by the key persons at Rembo Technology and is a predecessor of Rembo Toolkit. It has scripting support, in fact, many semantics in BpBatch appear in Rembo as Rembo-C functions. However, it lacks many features. There is no multicasting, no support for the NTFS filesystem, no registry patching features and no incremental imaging support. The possibility to create custom windows exist but are far less flexible compared to Rembo.

2.3.2 Norton/Symantec Ghost

Norton Ghost, developed by Symantec Corporation, is mainly marketed as an efficient backup tool for personal use. It provides support for NTFS and network stored images, but contains no remote boot facility. However, Norton Ghost includes a DOS-utility called Ghostwalker that can perform a small set of important post-installation registry modifications in Windows, such as changing the *Security IDentifier* (SID) and computer name settings. The SID problem is covered in [10] and in Section 3.2.6.

Symantec Ghost Corporate Edition [2] is a more modern cloning/imaging suite. Like Rembo, it is mainly targeted for enterprise use and supports both cloning of Windows and Linux partitions. The scalability shortcomings with its predecessor have been eliminated by applying multicasted and incremental imaging. Symantec Ghost uses a remotely deployed Ghost client on an invisible partition on each computer in order to provide space for Ghost's PC-DOS boot environment and network drivers. Thus, PXE is optional. The strength of Symantec Ghost lies in the Ghost Console, a way to remotely perform a predefined set of management operations.

2.3.3 Microsoft Sysprep

Microsoft Sysprep isn't really a cloning facility, merely a Windows tool supporting usage of dumb third party cloning tools with Windows 2000 [7]. As indicated

[6] This feature arrived with release 2.0 of Rembo Toolkit. The underlying mechanism employs MD5 checksums to index files [9].

by the name, it prepares the operating system for cloning by removing all computer specific settings. When an image prepared by Sysprep is restored and booted on a computer later on, Windows will repeat the configuration setup performed during the GUI part of the Windows installation. The setup can be carried out either unattended[7] or with user intervention via the normal installation wizard.

The method has a serious drawback though; The process of reconfiguration after restoring takes quite a while and may require subsequent reboots, thus rendering it useless for public computers typically synchronized on each startup.

Cloning Windows with Sysprep was initially evaluated and supported by SM RemboAdmin but was later abandoned. The intention has since then been to solve the same problems using the capabilities of Rembo, which have evolved significantly in this area. Some examples of how this has been achieved is discussed in Chapter 3.

2.3.4 The previous solution

Prior to migrating to Rembo the CSG used a home made image distribution solution, comprised of BpBatch, Norton Ghost and Ghostwalker. The solution applied to a computer lab of about 50 computers running Windows NT.

The PXE bootstrapped clients were configured to download a BpBatch executable along with a script to format and restore a small "superboot" MS-DOS partition at the end of the disk. This partition supplied network connectivity and a boot menu that either 1) started an already installed Windows NT partition or 2) invoked Norton Ghost from the MS-DOS partition. The latter option downloaded a raw, compressed image of Windows NT (~1 GB) over a unicast protocol, before Ghostwalker took control and patched the SID and various other registry entries. The system was now, typically one hour later, ready to reboot and run, after some additional registry modifications in the startup scripts of Windows had been made.

Needless to say, this solution wasn't very efficient, nor was it practical to maintain. The entire lab would typically take three hours to restore, during which the lab was unusable. Keeping the cloning source clean and intact was also a problem, not to mention the tedious work of modifying the superboot disk or the default image.

2.4 Project goal

The purpose of this work has been implement an application in Rembo, referred to as "SM RemboAdmin", that makes the everyday PC cloning tasks easier for the CSG. This holds particularly for configuring mass-distribution of disk images to a computer lab, although it should be able to make general desktop installation easier as well. The task of configuring and grouping single computers together is also an issue to avoid manual installation work. More specifically:

- *Multi-OS cloning.* The application must support convenient cloning of computers running Linux and/or one of Windows 98/2000/XP.

[7] Unattended in the context of software installation mean that the questions normally asked during installation are skipped. Instead of asking the user, all answers are stored in a text file.

- *Distribution of incremental images.* The main goal is to support convenient creation and installation of incremental images for Windows 2000/ XP, meaning differential snapshots of installed applications. It shall be possible to share such images in any combination among other computers with the same OS installed.
- *Group management.* When run, the application must provide a way to determine the host's group membership. Each group holds a number of settings, where the administrator should be able to configure partition layout, images to install and more.
- *Automated install and recovery tasks.* The application shall, if configured to, restore a base image and possibly a set of incremental images upon each reboot. It shall also be possible to protect partitions from being brutally overwritten, as well as allowing such actions for full recovery purposes.
- *OS customization.* The application must support the Windows OS:es in a set of personalization tasks after cloning. The most important are computer name changes, SID patching, hard drive layout registration and support for setting up machine accounts on a Windows NT domain controller.
- *GUI-controlled.* In addition, the application must provide a graphical user interface for the common tasks, typically manual image creation/ restoration/deletion, hard drive partitioning etc, as well as an interface for configuration.
- *Built for maintenance.* All work regarding the application must be thoroughly documented to support maintenance and further development independent of the initial author. The documentation should range all the way from installation to performing modifications on the application source code.

2.5 Limitations

At the time of writing, there are some tasks of the CSG where a solution isn't aimed at Rembo and SM RemboAdmin, even if reasonably possible.

- *No evaluation of alternatives required.* Rembo was already purchased and installed at the start of this project, the purpose with the project was merely to achieve the goals presented in Section 2.4.
- *No backup features.* The Rembo application is not required to perform disk-to-disk or network backups for permanent storage. For reasons of configuration time overhead, the present solution of a floppy booted version of Norton Ghost will be used instead.
- *No Linux/Windows 9x incremental images support required.* With the last Linux lab recently closed and only one Windows 98 lab left, no effort has to be put in creating Windows 9x incremental images. Regarding Linux (Solaris as well), applications have always been run from a *Network File System* (NFS). Rembo does not provide file system support for

the Windows 9x registry, yielding in no neat way to support incremental images. However, a possible solution to the incremental images problem with Windows 9x has been evaluated in Section 3.3.3.

Chapter 3

Cloning issues

With a background knowledge of the cloning scenario, this chapter continues with the identified customization problems when cloning the operating systems, their practical implications and what Rembo has in stock to facilitate this. Where relevant, the topics also include descriptions and discussions of tailor-made solutions implemented in SM RemboAdmin. The chapter also contains an evaluation of incremental images.

3.1 Cloning Linux

3.1.1 Base images versus remote installation

The task of maintaining Linux installations may differ a lot depending on the setup of the Linux distribution. The CSG has been using Red Hat Linux/Fedora for a number of years due to its level of hardware support and adaptation. This means that no additional customization is required after installation in the general case. One clone for each release is enough in most cases as new hardware is probed and supported via kernel modules upon boot. Thus, the cloning procedure in SM RemboAdmin limits to the setup of a general installation and emptying temporary directories with Rembo prior to cloning. In fact, any sector-based cloning utility would be sufficient for this task.

Since Rembo has the feature to boot Linux kernels with specified RAM disks and arguments, it is also possible to perform unattended Linux installations on demand. This has been practiced with the Red Hat/Fedora distribution using the tool Kickstart, making a complete re-installation merely a single click operation inside SM RemboAdmin. The kernel and RAM disk is located on the Rembo server, and the kickstart file supplied to the kernel resides on NFS mounted media.

Maintaining Linux hosts with Kickstart installations or cloning is basically a trade-off between the time required to perform the installations and the extra time required to maintain clones. Performing a Kickstart installation is significantly more time consuming than restoring an image. Depending on the amount

of clients involved and their re-installation frequency, one might also want to consider the advantages with multicasted image distribution in Rembo.

3.2 Cloning the Windows NT family

3.2.1 Base images versus remote installation

In contrary to Red Hat Linux discussed in the previous section, Windows NT and its successors are far less adaptable to hardware changes since device drivers more often are provided by third party vendors, making images less general. To make cloning less painful, it is considered a good idea to keep base images without installed applications. Focus should lie on maintaining the operating system patch level and supporting all hardware instead. With this in mind, a limited set of hardware groups with one base image each is recommended for each site implementing cloning.

As of Rembo Toolkit 2.0, released September 2002, creating base images of both FAT and NTFS partitions are possible. Previous versions were incapable of handling compressed NTFS folders (typically the Windows DLL cache) which required extra precautions.

By default, the NT family keeps the virtual memory in the hidden file `C:\pagefile.sys`. This file is of fixed size and defaults to about 200 MB, data which inevitably changes between each reboot and thus wouldn't be usable to the shared repository feature described in Section 2.2.2. By creating a virtual file system in Rembo during cloning it is possible to force exclusion of this file before uploading an image. This is done by default in SM RemboAdmin.

Rembo also has the feature to boot images of MS-DOS floppies and bootable FAT12 partitions as RAM disks, only limited in size by the available free extended memory. If such an image has TCP/IP and SMB support it is possible to use SM RemboAdmin to initialize an unattended installation of Windows only by booting the image. The booted partition will be responsible for invoking the installation using media residing on a SMB share. This is how the CSG normally wish to install the reference computer, that is the computer from which the base image will be created. However, it is very time consuming and requires sequential reboots. In addition, the procedure is often followed by installation of third party hardware drivers. Thus, cloning should be used instead whenever possible in order to avoid unnecessary work.

3.2.2 Incremental images

3.2.2.1 Overview

One of the main reasons for implementing a Rembo solution was to be able to benefit from incremental imaging. While one base image is needed per hardware group, incremental imaging ideally makes it possible to share installed applications by applying them independent of each other on any Windows NT system on top of any base image. Both the concept itself and using the API functions to implement it is really straightforward. However, using it blindly comes with a lot of surprises which is why it is examined in detail in order to pose a discussion about its suitability.

Incremental imaging support in Rembo is implemented as a wrapper on the usual disk imaging functions, achieved by creating virtual images of the current disk content and a base image in order to extract the differences, omitting the registry files. The registry files are then mounted and compared separately in the same fashion. The entire procedure is twofold; first off a file containing all differences is created, allowing for manual review of the prospective image content. Secondly, all data is uploaded to the server as described in the list of differences. However, as files are compared by time stamps[1] in order to detect changes, an incremental image will most likely contain more changes than necessary.

While incremental imaging makes installation of a single computer an easy task, provided all applications are already packaged, it must be pointed out that creating these images is a time consuming task. The system must be restored to the base image, booted, followed by the software being installed. Additional configuration and subsequent reboots may also be required before the image can be created. Prior to being used in production it should also be tested on other computers, perhaps with other versions of the operating system before applying it in production. Furthermore, it should also be tested in conjunction with other incremental images to make sure no undesired behaviors occur. The entire procedure must be repeated for each new version of the software, although some of the testing can be omitted. With this in mind, consideration should precede the making of incremental images with respect to usage and update frequency, sharing benefits and required time for configuration. It might just as well be sufficient to supply the application in the base image or install it manually.

3.2.2.2 Dependencies

The experience from working with incremental images is that it can really be both a blessing and a curse. While many applications cooperate flawlessly, some refuse to move between hosts/base images and might even make other software unusable. As an example, Symantec Antivirus seem to be a common troublemaker when applied incrementally on another Windows installation. It is known to destroy both Microsoft Office and some basic Windows functionality. However, it lies in the nature of this software to act hostile against unexpected changes.

Applications with licenses tied to a specific host, typically using product keys during installation, has also been troublesome in some cases with incremental imaging. The intention of the programmer is of course to prevent copying of the installation directory. With site licensed software it is still desired to avoid the manual installations. A common solution to work around this problem is to include this software in the base image.

Attention must also be paid to dependencies between Windows applications. A good example would be Adobe Acrobat which installs a plug-in in Microsoft Office to support saving documents in PDF format inside Office. When packaged alone from a base image containing Office, Acrobat will complain if Office isn't

[1] Rembo is also able compute MD5 digests on all files to detect changes in a more secure manner. The MD5 comparison mode is slower than the time stamp mode, because it requires Rembo to read the content of the destination files for comparison with the MD5 digests stored in the server archive.

present. Another example; Java Runtime Environment (JRE) adds a plug-in to Mozilla to provide applet support. As these applications more or less require each other, it is considered a good idea to include them in the same incremental image. If they are to remain separated, create the images on a really clean base image.

3.2.2.3 Reversibility

Another issue not handled by Rembo and SM RemboAdmin at the moment is a way to maintain installed incremental images on a computer. A list of installed images and the possibility to remove specific images is desirable. Removing images would require backup features on the server for each file and registry key overwritten and is not planned to be implemented. Performing uninstallation is currently only possible the standard way, either from the Control Panel inside Windows or from a supplied uninstallation program.

3.2.3 Changing computer names

Unlike most other operating systems, Windows NT does not automatically use the hostname supplied by the DHCP server. Besides the regular DNS hostname, Windows also maintain a NetBIOS name [8] in order to facilitate "Windows networking", i.e. file sharing over the SMB protocol. NetBIOS has a special naming scheme kept for historical reasons, in which clients choose their own names on a first come first served basis. Name lookup is done via broadcast or a special WINS server, a DNS equivalent for NetBIOS names. When a Windows client detects a NetBIOS name collision on the network it immediately disables all functionality related to Windows networking.

Rembo comes with API support for writing an arbitrary hostname and NetBIOS name to the registry using two different functions. Since the DHCP server response is accessible inside Rembo, both names can be updated in the registry with the name in DNS before each boot to avoid collisions. Furthermore, this action prevents invalid names from getting duplicated on the network when applying incremental images as each such image will contain the names of the reference computer.

3.2.4 Joining a Windows NT domain

3.2.4.1 Domain purpose

A common way to provide a user database with authentication for Windows NT family workstations in multi user environments is by means of a NT domain, hosted by the *Primary Domain Controller* (PDC). In order for a computer to authenticate users against the PDC, the NetBIOS name of the computer must have a *trust account* in the PDC. A trust account is like any user account, named from the NetBIOS name followed by a dollar sign ($). Furthermore, the host and the PDC must agree on the trust account password as this will be verified when accessing the PDC. Normally the domain administrator user has to authenticate from the host in order to join the domain, so this has to be done prior to cloning.

Inevitably, this is a case where the Rembo registry modification capabilities come in handy as the domain information and trust account is stored in the

registry. However, the process can unfortunately only be fully achieved by the Rembo API if the server is also running the Microsoft *Active Directory* (AD) catalogue service. AD is a modern, more diverse solution than a regular Windows NT PDC. Since the CSG is running Rembo on Solaris, using Samba with a LDAP backend as PDC, a customized solution has been implemented in SM RemboAdmin.

3.2.4.2 The domain joining procedure

The process can be divided into two different tasks. First off, the domain settings, if available, must be read from the registry in order to generate a new trust account password and store it in clear text inside the host registry. At the same time a Windows executable called `joindom.exe` supplied by Rembo is copied to disk and installed in the registry as a service with the purpose to encrypt the clear text password. The service is removed once it has run in Windows during boot. This part of the solution is a modified version of the code in the Rembo API function `NTJoinDomain()`.

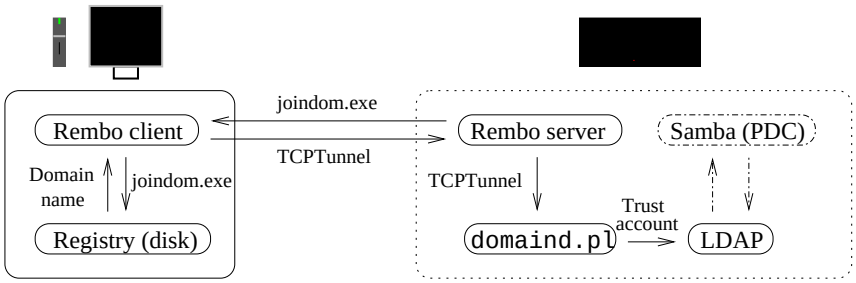

Figure 1: The connections between different services in the domain joining procedure.

The second part comes down to contacting the LDAP server in order to create/set the trust account with the generated password. This takes place between completion of the image transfer and client boot. Figure 1 illustrates the process. As Rembo doesn't support communication via the LDAP protocol, the task of adding the trust account has to be done manually. We achieve this by setting up a TCP tunnel on the client via the Rembo server to a specific port on the server. The tunnel is responsible of transmitting the domain name and trust account information. On the server an `inetd`[2] service is installed, configured to run a Perl script called `domaind.pl` (included in SM RemboAdmin). The script is responsible for connecting to the LDAP server to create or modify the trust account. The Samba server is never contacted in this scenario, hence the dotted arrows in the figure. Note also that the Rembo, `domaind.pl`, LDAP and Samba services can reside on four different machines, Figure 1 only reflects the CSG scenario.

[2]We use *Internet services daemon* (inetd) to execute processes upon connections on certain ports. It also has the desired feature to supply incoming TCP data on the port to the standard input of the spawned process, which is how credentials are transferred between the Rembo client and the LDAP server via `domaind.pl`.

3.2.4.3 Domain security issues

The idea with trust accounts is to prevent unauthorized computers to use the domain resources. Looking at the solution above, it would be trivial to create arbitrary trust accounts by just looking at the source code of SM RemboAdmin and send some data to the `inetd` service listening for connections from the TCP tunnel. This situation has been taken into consideration and can be avoided in a number of ways.

First of all, make sure the port for the TCP tunnel is open only for connections from the Rembo server, i.e. localhost in our example. This can easily be achieved with the `tcpd` program on Unix systems. The purpose of `tcpd` is to act as a wrapper between `inetd` and the service to be executed, providing logging and additional checks such as IP restrictions prior to any communication with the service.

Secondly, it is desired to control which clients should be allowed to create trust accounts. This has been implemented with a NIS map maintained on the server[3], in which all accepted domain members are listed. This is where the `domaind.pl` script will look for hostnames prior to updating any accounts in LDAP. The solution is relevant and works as the NetBIOS name and the name of the corresponding trust account cannot differ. It protects mainly against Rembo clients not allowed to join the domain, but also any malicious user in the system if the Rembo server is open for remote logins.

However, there are still some problems unsolved regarding to security. The initial trust account password is transmitted in plain text between client and server. This could be improved somewhat as proposed in the Rembo Toolkit Manual [9] by adding symmetric blowfish encryption, preferably in conjunction with MD5 sums on both ends. This however introduces a key distribution problem, although it would provide some protection from a casual eavesdropper if implemented. Furthermore, verification of the connecting host's name inside `domaind.pl` should be inserted to prevent possible misuse of other's trust account.

3.2.5 Hard disk drive registration

In order to keep track of hardware changes, Windows stores information tied to the installation in the *Master Boot Record* (MBR) of the hard disk drive. Windows also stores drive specific information such as vendor and model name along with the serial number in the registry. When Windows is booted after cloning, it immediately detects the hardware change since the serial number in the registry mismatches the one on disk and thus it wants to modify the registry and reboot. This is a common first mistake in production and a hassle found quite annoying by users.

However, ignoring this issue does not come without pitfalls. It is often desired to synchronize hard drive content with the base image in Rembo before starting Windows, and thus the detection of new hardware will appear upon every boot. Furthermore, the registry update occurs after logging in, making an unprivileged user prohibited of applying the required changes. In addition, Windows sometimes tend to silently skip the launch of the crucial service *Net*

[3] Although not relevant for this discussion, automatic maintenance of this feature is discussed in Chapter 4.

Logon required to authenticate users with the NT domain. Thus, the detection of new hardware may prevent users from logging on to the machines.

The Rembo Toolkit has a pair of API functions to help solve the registry related problem. In addition, it is required to store the MBR while producing base images for later restoration. Both problems are fixed transparently in SM RemboAdmin. It also performs the registry update after applying incremental images in order to support several cloning machines.

3.2.6 Changing SID

A *Security IDentifier* (SID) is a unique alphanumeric string used to identify users, groups and computers in the Windows NT family. The user SID is generated either from the computer SID or the domain SID, depending on if the user has been created locally on the machine or in a Windows NT domain. Each time a user logs on, an access token containing the the user's SID and group SID:s is created that will be attached to the processes of the user. In addition, SID:s are stored in the *Access Control Lists* (ACL) of the NTFS file system to support file ownership. To ensure unique user SID:s, each user has a *Relative IDentifier* (RID) concatenated at the end of the computer/domain SID. The RID is basically an integer similar to the Unix UID[4]. The same holds for group SID:s.

Security is the main reason for providing each computer with a unique SID. When users are authenticated against a PDC all users are generally made unique as their RID:s differ. However, in the case of a set of cloned computers with local users in a workgroup environment, there is no possibility to distinguish between users having the same RID as the computer SID:s will all be the same. In particular, the Administrator user always has the RID 1000, making such an environment dangerous [10]. This problem is also an issue with removable NTFS media. Thus, it is desired to create a new computer SID and replace all occurences of the old one after a base image has been written to disk.

Rembo provides API support for extracting SID:s from the registry or disk images and generating new SID:s in order to replace all occurrences of the old SID in the registry. In addition, a special function `ApplySID()` exists to make sure all occurrences of the SID in ACL:s in the disk image is replaced with the new SID when an image is written to disk.

In SM RemboAdmin the change of SID is optional but encouraged, in particular for office workstations as they normally aren't part of the domain. However, computers tend to enter and leave the domain for practical reasons over time. The first time a specific host (distinguished by the MAC address) is cloned, a new SID is generated and stored on the Rembo server. This way the same SID can be restored each time the computer is synchronized, which is very important if the user has local data on a partition different from the one being synchronized.

SM RemboAdmin also has a GUI for viewing the current state of the SID:s stored on disk, in the base image and on the server, along with the feature to remove the server SID. As the SID length and syntax varies between Windows

[4]The RID r can be constructed from the Unix UID u as $r = u * 2 + c$, where $c = 1000$ for an user RID and $c = 1001$ for a group RID. This is a Samba specific feature called *Algorithmic RIDs*, and is very convenient in a mixed Unix/Windows environment, using Samba as a PDC with the user database in a LDAP directory.

2000 and XP it has to be removed on server and regenerated if a host changes operating system.

As with other registry based Rembo operations it is very important that the machine used to produce incremental images does not patch the SID each time the machine is cloned, as these changes would appear in the incremental image. Applying such an image might render the system in question unusable.

3.3 Cloning the Windows 9x family

3.3.1 Base images

In contrast to the NT family (Section 3.2), Windows 9x comes totally without surprises when distributing base images. It has no nasty behaviors with changed partition tables and hard drive serial numbers, and since it lacks the feature to join a Windows NT domain, no support is required for it. Furthermore, the Windows 9x family contains no SID:s. Thus, the cloning task consists of merely changing the hostname (described in Section 3.3.2) and saving/restoring the MBR.

3.3.2 Patching the Windows 9x registry

In Windows 9x the registry consists of three binary files named `SYSTEM.DAT`, `USER.DAT` and `POLICY.POL`, located in the Windows directory, e.g. `C:\WINDOWS` [6]. The first two are mandatory and contains computer-specific settings, that is the hardware profile and the user profiles, respectively. The third is optional and can be used to override settings in the other two. As already pointed out, Rembo provides no support for modifying the content of these files, and neither can string operations be applied due to their binary format.

The solution is limited to using Windows' own registry editor `regedit.exe`. Regedit is used to browse and edit the registry, but can also import and export text representation of the keys and values. The latter is a feature that also can be run in DOS mode, thus our solution when solving the hostname problem (introduced in Section 3.2.3) involves patching a text file containing a set of registry entries. The remaining task is to make sure Regedit is run on the patched text file upon startup from `C:\autoexec.bat`. This is how registry keys, e.g. the computer name(s) are patched into the registry. The text representation of the registry keys are typically generated and written to disk by SM RemboAdmin prior to booting Windows.

3.3.3 Incremental images

Based on the discussion in the previous section, creating and restoring incremental images in Windows 9x adds a lot of work in terms of handling registry modifications made by the applications during installation as there is no way to merge the registry changes as supported by Rembo for Windows NT (Section 3.2.2). In order to evaluate the possibilities, one attempt has been made to produce an incremental image of a large application. It turned out successful, but due to the procedure involved it is not considered a cost effective solution.

1. Install and boot a base image. Use Regedit and export the entire registry (`HKEY_LOCAL_MACHINE`) to a file.

2. Run the installer for the desired application and export the registry afterwards to a different file.

3. Use a text comparison tool to extract the difference between the files in (1) and (2) with the result stored in a new file. The Rembo Manual [9] suggests the MS-DOS utility `fc`, but it appears a more competent tool is required.

4. Create an incremental image in Rembo as usual. This image will include the file produced in (3).

The deployment procedure would then be to apply the incremental image, followed by patching the registry as described in Section 3.3.2. The solution is rendered infeasible due to the fact that step (3) has to be checked manually, making it time consuming and error prone.

Chapter 4

Custom-built concept

Given the customization problems and solutions covered in the previous chapter, this chapter brings forth how the SM RemboAdmin can be configured to solve the problems. It gives a practical view both of how computer grouping as well as the provided interface for configuring group features work.

4.1 Computer identification and group membership

4.1.1 Maintenance of static hosts with HOSTGEN

The CSG has been using the DHCP protocol over the last decade to statically deliver IP addresses to all clients in the network. One feature with static IP addresses is that each address is tied to a specific computer, or in practice the MAC address of the network interface. Not only does static addresses make it easier to keep track of clients in server logs, but it also gives us aid in grouping computers together depending on their hardware configuration and software requirements. As different base image and other Rembo related settings are needed for each hardware configuration and target usage, the groups are defined as a set of computers sharing the exact same options inside SM RemboAdmin. In this section the group maintenance aspect is described.

For the last couple of years, the CSG has been using a home made host database tool called HOSTGEN[1], to allow for convenient configuration of most network services requiring per-host information. At this date the involved services are DHCP, DNS, LPRng (printer spooler system), Solaris Jumpstart installations and Rembo. The database consists of semicolon separated lines, with one line per host containing with all relevant information for these services as shown in Table 4.1.

[1]HOSTGEN is a growing set of `sh`/`awk` scripts with some `C` optimizations. It was originally designed and implemented at the CSG by Assar Svensson and Per Björn.

```
staff-pc-linux;00:90:27:bd:8e:79;130.240.3.220;taumaster.csee.ltu.se; \
   Johan Hallbäck;Linux cloning test;hallback@csee.ltu.se;stafflinux
```

Table 4.1: The HOSTGEN entry of a Rembo client.

The fields in Table 4.1 have the following properties:

1. `staff-pc-linux`: This field denotes the class of the current host. Each class determines a set of network services to configure in favor of the current machine. An example of use could be whether HOSTGEN should add configuration entries to set up Solaris Jumpstart. For Windows hosts we might use a class that configures NIS to provide the possibility of having the host as a NT domain member, as discussed in Section 3.2.4.

2. `00:90:27:bd:8e:79`: The MAC address of the network interface in the host. Not only is it required for static DHCP, but HOSTGEN also needs it when generating `rembo.conf` (the configuration file for the Rembo server) in order to make the host bootable via Rembo.

3. `130.240.3.220`: The IP address of this host.

4. `taumaster.csee.ltu.se`: The *Fully Qualified Domain Name* (FQDN) of the host.

5. `Johan Hallbäck`: The host owner.

6. `Linux cloning test`: A field normally used for miscellaneous info such as department, office number, machine description etc. Invaluable when tracking down machines from log files. This field is added as a comment in the configuration file(s) of each service set up for this host.

7. `hallback@csee.ltu.se`: The e-mail address to the responsible administrator of this host.

8. `stafflinux`: The Rembo group membership identifier. Each unique string in this field form a group in SM RemboAdmin which will be used to determine cloning settings.

The upcoming subsections will show how group control is transferred to SM RemboAdmin and how this can be used for customization.

4.1.2 The booting process

While the intention is to leave the most gory details of the DHCP [3] and PXE [5] protocols out in this document, this section describes some basic functionality in DHCP. It will be shown how HOSTGEN can help in configuring DHCP to transfer the group membership identifier (HOSTGEN option #8 in Section 4.1.1) to SM RemboAdmin.

The DHCP standard defines a large set of options in order to supply clients with network configuration parameters such as IP addresses, DNS servers etc., but also allows the user to define new options. Each DHCP option has a name, a number, a code and type. Option codes 128 to 254 are reserved for site-specific

options [1]. For this purpose code 135 has been declared as `option-135`, a text field containing the group membership identifier.

The entire booting process including group identifier transmission is detailed in Figure 1. Prior to any PXE boot server discovery, the client broadcasts a DHCPDISCOVER request (Figure 1:1) on UDP port 67 in order to get an IP address. The request also contains the DHCP option PXEClient (#60), pointing out that the client is ready to receive a list of boot servers which may or may not be received in the unicasted DHCPOFFER response (Figure 1:2) from the DHCP server(s). The client will acknowledge the new IP to one of the servers (Figure 1:3) and get it confirmed (Figure 1:4). Only the extra PXE flags and options differ from a normal DHCP negotiation.

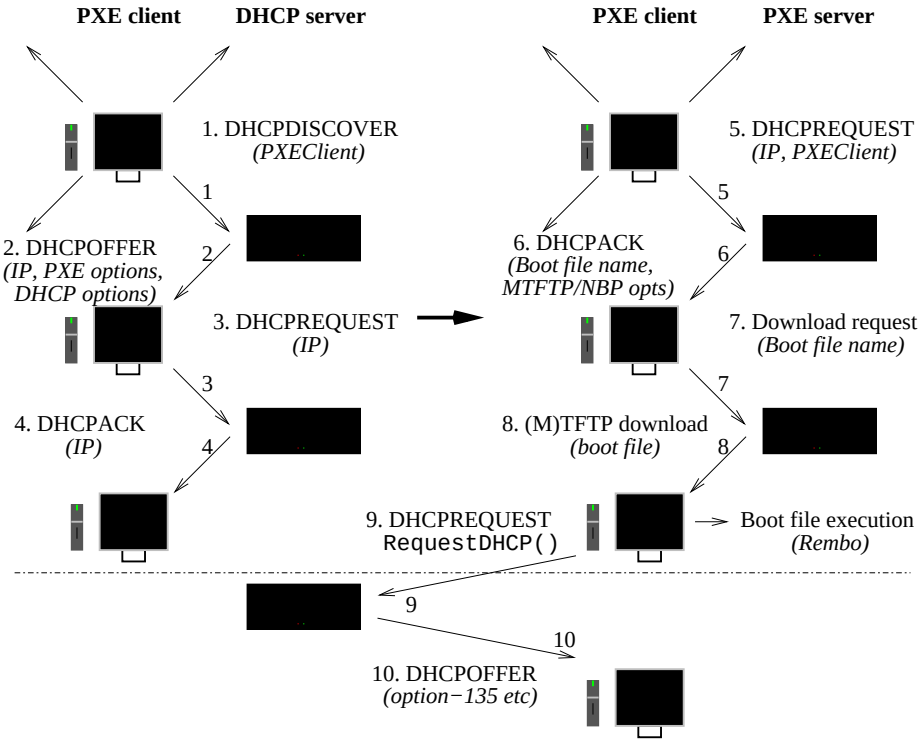

Figure 1: The PXE extended DHCP request (left) followed by the PXE boot server discovery.

At this moment the Rembo server gets involved acting as PXE server, shown in the right part of Figure 1. The client broadcasts[2] a DHCPREQUEST (Figure 1:5) which is like the DHCPDISCOVER request from step 1 but also containing the received IP address. As we choose only to allow known MAC addresses to run Rembo, the Rembo server will verify that the MAC address in the DHCPREQUEST packet is present in the HOSTGEN generated file `rembo.conf`. The first boot server to reach the client with a DHCPACK response (Figure 1:6)

[2]Note that this DHCPREQUEST may be unicasted if the DHCPOFFER in step 2 contained a boot server. The CSG use Rembo to allow clients to boot via MAC address identification instead of selecting boot servers using the PXEClient DHCP extensions.

containing a boot file name and additional options (such as download port) will win the client and prepare for the client to issue a download request (Figure 1:7) using the *Multicast Trivial File Transfer Protocol* (MTFTP). The *Network Boot Program* (NBP), the Rembo environment in our case is, is now loaded (Figure 1:8, 9).

As a last operation in the Rembo startup it downloads and executes a predefined Rembo-C script, which in turn will check a settings file for further execution instructions. This is how the SM RemboAdmin precompiled executable is run. However, despite the fact that all DHCP options was retrieved in step 2 of Figure 1 there is no possibility to obtain their values as this data was acquired prior to the execution of Rembo. This makes yet another DHCPREQUEST (Figure 1:9) the first task of SM RemboAdmin in order to retrieve the `option-135` string (Figure 1:10) needed to determine group membership.

It may be argued that `option-135` could have been omitted since an equivalent string can be retrieved inside Rembo by having MAC addresses sorted into `GROUP` clauses inside `rembo.conf`. Such an option would however have prevented the SM RemboAdmin grouping concept on a Rembo server configured to boot all clients.

4.2 Configurability and interface

4.2.1 Configuration options

A parser module in SM RemboAdmin is run in order to determine the group configuration parameters after the client has acquired its group identifier via DHCP. If `option-135` is missing the membership defaults to group `other`. All group settings are located in the same text file on the server (`scripts/SM_RemboAdmin.conf`). The file can be edited manually either on the server or on an authorized client, but the supplied graphical configuration tool mentioned in Section 4.2.2 is recommended to maintain correct syntax and values. The syntax is shown in Table 4.2.

```
# Comment
[stafflinux]
  linux = true
[other]
  wintype = winnt
  patchsid = true
```

Table 4.2: An example of the configuration file.

In order to get a better grip on the features and configurability of SM RemboAdmin, this section briefly introduces some of the options a group can take on and their meaning. The values are either boolean (`true` or `false`), or any strings with or without syntax checks. Syntax checking means that the parser will complain if such an option has a value different from a predefined set.

wintype (string): Indicates if the host should run Windows or not. Valid values: win9x|winnt|none.

linux (boolean): Indicates if the host should run Linux or not.

autoboot (string): Indicates action (boot, reinstallation or none) to be performed automatically by a client when SM RemboAdmin is launched. Valid values: win9x|winnt|linux|reinstall|none.

fullinstallmethod (boolean): Determines if the hard drive should be formatted (full install) or not prior to synchronizing the base image to disk.

partitions (string): A text representation of the partition table to create prior to writing base images in full installation mode.

defaultwinbase|defaultlinbase (string): The name of the base image to use during reinstallations of Windows and Linux, respectively.

defaultkernel|defaultramdisk (string): The full URL of the kernel and RAM disk to load (respectively) when booting Linux.

installkernel|linuxramdisk (string): The full URL of the kernel and RAM disk to load (respectively) when performing an unattended installation of Linux (Section 3.1.1).

windowsramdisk (string): The full URL of a MS-DOS disk image to use for unattended installations of Windows 2000 (Section 3.2.1).

blankdosfloppy (string): The full URL of a formatted MS-DOS system floppy used to merge with a disk tree to create the `windowsramdisk`.

kernelcommand (string): The command to supply the kernel with when performing an unattended installation of Linux. Typically the location of a Kickstart file on a NFS server.

linuxrootdevice (string): The value for the `root` argument to be passed to the Linux kernel at boot time.

preservepartitions (boolean): Indicates if the hard drive content should be protected or not from alteration during reinstallation if the partition table differs from the `partitions` option.

patchsid (boolean): Indicates if the SID should be patched or not, as described in Section 3.2.6.

joindomaininclone (boolean): Indicates if the NT domain should be joined or not, as described in Section 3.2.4.

doublepartitions (boolean) Indicates if the group has a second disk image or not to be applied on the second primary partition.

showsystemrestore (boolean) Indicates if anonymous access should be given, or not, to restoring a computer to the state of its base image.

Despite the fact that adding new options to be used by the groups are fully possible, the parser module currently has no way of coping with options for SM RemboAdmin itself. An idea for future development is to follow the Samba configuration file semantics and have a section called `[global]` where the generic settings are located. Such a section could have several options:

- A way to set a custom name on `option135`
- Possibility to adjust the timeout duration when the `autoboot` option is set
- Set any maximum boundary on the size of the `windowsramdisk`

Although this is not critical, there are a few places in the code where reasonable constants have been applied.

4.2.2 User interface and features

After the group options have been parsed the SM RemboAdmin GUI is launched. All functionality is located as icons on the interface desktop, accessed with a left-click on the mouse. A right-click on the mouse on any item provides a help text describing the impact of the operation, along with all relevant SM RemboAdmin options and their current values. The interface has been divided into two different menus for security reasons. The boot menu, shown in Figure 2, contains the following:

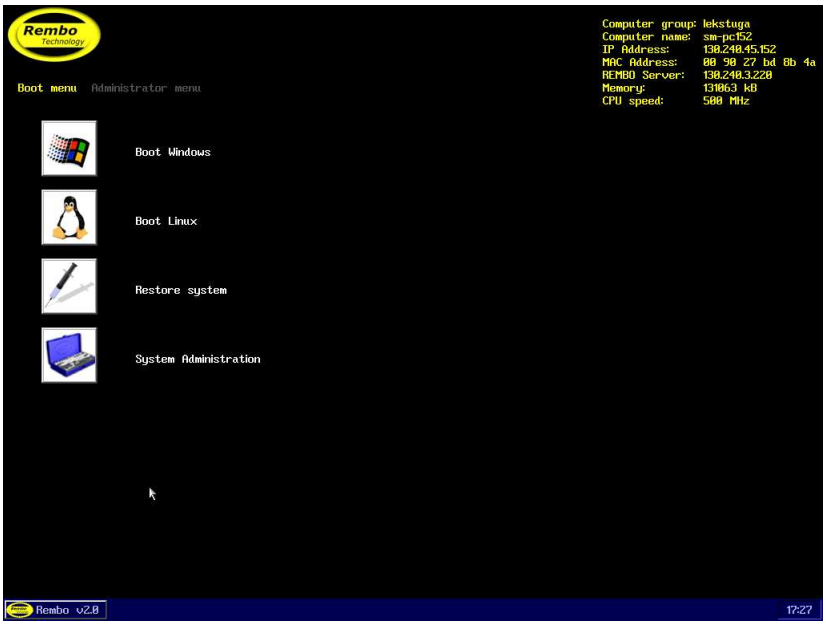

Figure 2: The boot menu.

- *The boot loader*, showing an icon for each operating system specified in the configuration. The Linux part of the boot loader is responsible for loading the correct kernel/ram disk according to the configuration parameters. The Windows part makes the hostname related registry modifications (Sections 3.2.3 and 3.2.5). The boot loader might also be activated automatically after a 15 second progress indicator timeout as specified by the `autoboot` option.

- *System restore* (optional). For some groups, typically computer labs, it might be suitable to give unauthorized users the possibility to synchronize the base image to disk if the computer appears to have been tampered with. On the other hand it might be dire if accidentally performed on a machine with valuable local data. The behaviour is determined by the `showsystemrestore` option.

- *Administrator authentication.* The link to the administrator menu is password protected. In order to access it the user must be queried for valid credentials, determined and verified by the server.

Figure 3: The administrator menu.

The administrator menu in Figure 3 contains the following:

- *Auto-layout hard drive.* This module gives a view of the disk size and current partition table. It also gives the possibility to repartition and format the drive according to the table specified in option `partitions`.

- *Partition editor.* This button gives access to the graphical partition editor supplied with Rembo Toolkit, allowing for manual setup of the disk layout.

- *Manual image installation.* This module allows the user to choose base images in the group scope for the configured operating systems. In addition, it is possible to choose any set of incremental images available for the current Windows version, determined by option `wintype`. After selection, all involved partitions are formatted and the images are applied sequentially.

- *Netinstall Windows.* This module boots the `windowsramdisk` image, allowing for initialization of an unattended Windows installation (Section 3.2.1).
- *Create RAM disk from local drive.* Provides a file requester in order to merge a directory tree on local disk with the `blankdosfloppy` in order to create the `windowsramdisk` file. The module can also create the `blankdosfloppy` from a formatted floppy drive if the file is missing.
- *Netinstall Linux.* Boots the `installkernel` with the `linuxramdisk`, providing the `kernelcommand` for an unattended Linux installation (Section 3.1.1).
- *Create images.* Provides an interface for creating and naming base images and incremental images.
- *Delete images.* Allows for removal of base images in the group scope and incremental images in the global scope.
- *Configure SM RemboAdmin.* To achieve the project goal of GUI configurability introduced in Section 2.4, this function is a graphical wrapper for the previously mentioned parser module. It provides an interface for editing all configuration options (listed in Section 4.2.1) for all available groups. The boolean options are represented as checkboxes for ease of use, while the others are shown as text fields, illustrated in Figure 4. All values are verified when saved on server.
- *Remove host-specific SID.* This module shows the current SID status, i.e. it shows the SID:s in the `defaultwin` base image, in the registry on disk and on the server. The purpose is to allow for removal of the host SID located on server for reasons discussed in Section 3.2.6.

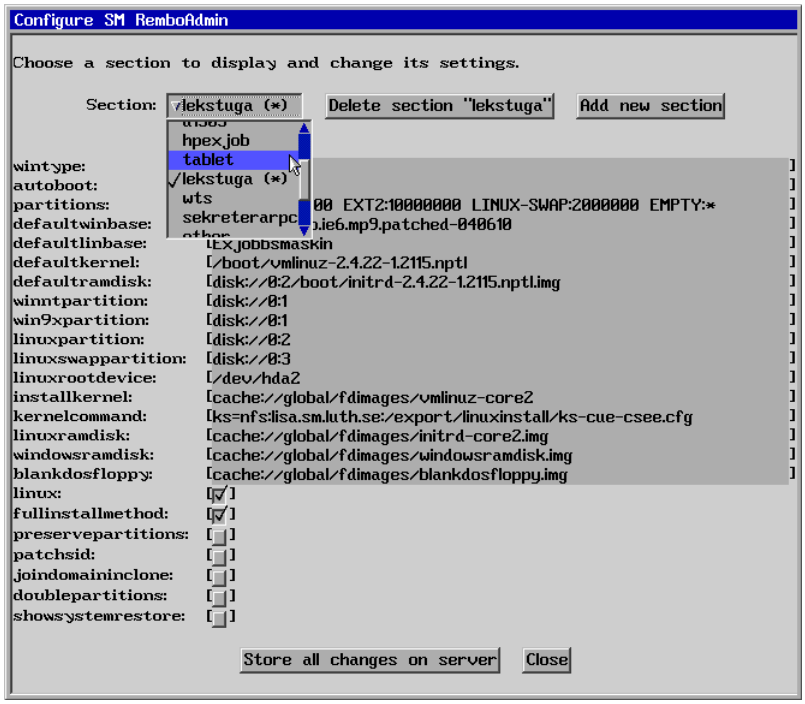

Figure 4: The SM RemboAdmin configuration tool.

Chapter 5

Conclusion

In this final section the key discussions from the previous sections are summarized along with some ideas for future work.

In this thesis some problems and benefits with cloning of contemporary operating systems have been presented. The focus has been to show the possibilities and features with a Rembo Toolkit implementation called SM RemboAdmin created during the work of this thesis. While the main reason for launching this project was to benefit from incremental imaging - the concept of applying pre-installed software on a Windows installation, the implementation phase showed other aspects of equal importance which also were treated in this thesis. In particular, customizing operating systems of the cloned clients (i.e. making them unique) and configuring groups of similar clients for scalability benefits has also been taken into consideration.

With the cloning tool Rembo Toolkit already purchased at the start of the project, the task was limited to achieving the goal of a maintainable, configurable GUI controlled administrator interface defined in Section 2.4. In Chapter 3 we presented the issues with cloning Linux and Windows, focusing on the latter (in particular the Windows NT family). We described the generality with incremental imaging, but also its drawbacks when software is tied to a particular computers. Other issues such as software dependencies, time for image creation and testing plus the lack of an uninstall function were also identified. The overall conclusion regarding incremental images is that the solution in many cases can be used to simplify the deployment, however, it is not fully solve the application distribution problem.

Chapter 3 also treated some security aspects. We addressed the SID problem in Section 3.2.6 solved by the features of Rembo toolkit, while the domain joining problem required a tailor made solution presented in Section 3.2.4. The latter problem could gain from future work by encrypting the trust account credentials that are now being transferred in plain text.

Chapter 4 focused on the features of SM RemboAdmin and how it solves the computer grouping problem with use of the DHCP protocol and the home

made host management tool HOSTGEN. The attempt was to show the Rembo solution's advantages in a standard services environment, also proven with the previously mentioned domain joining solution. In addition, possible configuration options a computer group can take on were introduced to give an idea of the SM RemboAdmin features, followed by an overview of the graphical interface and the configuration tool.

As with all software, the maintenance and upgrade cycle should go on. Throughout Chapter 4 we identified areas where further development could improve SM RemboAdmin. In particular, the application itself should be able to configure, not only the computer groups. Support for more operating system combinations would also be desirable, typically combinations of different Windows versions on a single computer.

Bibliography

[1] S Alexander and R Droms. RFC 2132 - DHCP Options and BOOTP Vendor Extensions. Silicon Graphics Inc./Bucknell University, March 1997. http://www.ietf.org/rfc/rfc2132.txt.

[2] D Dem. Managing Disk Partitions Over the Network with Ghost Corporate Edition. Pearson Education, Que Publishing, October 2003. http://www.quepublishing.com/articles/article.asp?p=101647.

[3] R Droms. RFC 2131 - Dynamic Host Configuration Protocol. Bucknell University, March 1997. http://www.ietf.org/rfc/rfc2131.txt.

[4] D Esposito. Windows 2000 Registry: Latest Features and APIs Provide the Power to Customize and Extend Your Apps. Microsoft Corporation, November 2000. http://msdn.microsoft.com/msdnmag/issues/1100/Registry/default.aspx.

[5] Intel Corporation. *Preboot Execution Environment (PXE) Specification*, 2.1 edition, September 1999. ftp://download.intel.com/labs/manage/wfm/download/pxespec.pdf.

[6] Microsoft Corporation. Description of the Registry Files in Windows 98/95, December 2000. http://support.microsoft.com/default.aspx?scid=kb;en-us;250410.

[7] Microsoft Corporation. Using Disk-Image Copying in Microsoft Windows Deployment, January 2001. http://www.microsoft.com/technet/prodtechnol/ntwrkstn/deploy/depopt/clo%ning.mspx.

[8] M Minasi. NetBIOS names and WINS. Windows IT Pro, January 1997. http://www.winnetmag.com/Windows/Article/ArticleID/117/117.html.

[9] Rembo Technology SaRL. *Rembo: A Complete Pre-OS Remote Management Solution - Rembo Toolkit 2.0 Manual*, August 2002. http://www.rembo.com/rembo/docs2/rembo/.

[10] M Russinovich and B Cogswell. Utilities for Windows NT and Windows 2000: NewSID. Sysinternals Freeware, February 2003. http://www.sysinternals.com/ntw2k/source/newsid.shtml.

Appendix A

List of abbreviations

ACL: Every object in Windows NT has a unique security descriptor that includes an *Access Control List*. An ACL is a list of entries that grant or deny specific access rights to individuals or groups on a resource.

API: An *Application Program Interface* is a series of software routines and development tools that comprise an interface between a computer application and lower-level services and functions, such as the operating system, device drivers and other low-level software. API:s serve as building blocks for programmers putting together software applications.

BIOS: The *Basic Input/Output System* is the root software in a PC that contains all of the basic code for controlling drives, keyboard, monitor, mouse, serial ports, etc. The BIOS acts as a bridge between the hardware and the operating system.

CIFS: The *Common Internet File System* protocol is a successor of the SMB protocol, and runs on top of TCP/IP as the native file sharing protocol in modern versions of Microsoft Windows. CIFS has been designed with the intention to improve on concurrency, security and performance.

DHCP: *Dynamic Host Configuration Protocol* is a server-client protocol for network distribution of IP-addresses and associated network configuration parameters.

FQDN: The *Fully Qualified Domain Name* is the complete address of a host, e.g. `hostname.some-domain.net`.

LDAP: Short for *Lightweight Directory Access Protocol*, a set of open protocols for accessing information directories over TCP/IP. LDAP both an information model and a protocol for querying and manipulating the directory service.

MAC: Short for *Media Access Control*, the lower of the two sublayers that make up the Data Link Layer of the OSI model, i.e. the interface between the link control and the physical layer of a shared network. The MAC address is the unique hardware address of a network interface.

MBR: The *Master Boot Record* is the first logical sector of a hard drive. This is where the BIOS will look for information to determine where the operating system is located on disk and how to load it.

MTFTP: The *Trivial File Transfer Protocol* is a simple file transfer protocol typically used for downloading boot code to diskless workstations. The multicast variant MTFTP is used by Rembo when transferring the remote boot operating environment to the client.

NBP: When booting from the network using PXE, the system or network adapter BIOS uses DHCP to locate a *Network Bootstrap Program* on a boot server and reads it using (M)TFTP. The BIOS executes the NBP by jumping to its first byte in memory.

NFS: The *Network File System* was originally developed by Sun Microsystems in the 1980's as a way to create a filesystem on diskless clients. NFS provides remote access to shared filesystems across networks.

NIS: The *Network Information Service* is a distributed network database, typically containing users, groups and computers in a network. Originally called *Yellow Pages* (YP) until forced to change name due to trademark infringement on British Telecom.

NTFS: *New Technology File System* is the file system being used in the Windows NT. NTFS is required if access control should be used on files in Windows.

PDC: *Primary Domain Controller* and *Backup Domain Controller* (BDC) are roles that can be assigned to a server in a network of computers that use operating systems in the Windows NT family. Windows NT uses the idea of a domain to manage access to a set of network resources (applications, printers, file shares and so forth) for a group of users. The user need only to log in to the domain to gain access to the resources.

PXE: *Preboot eXecution Environment* is an industry standard protocol developed by Intel Corporation to use a network interface as boot device for operating systems.

SID: The *Security IDentifier* is a unique name (alphanumeric character string) that is used to identify users, groups and domains in a Windows NT system. The SID is stored in the ACL:s of a resource to allow or deny access to the particular resource.

SMB: *Server Message Block* is an open, cross platform protocol for sharing files, printers, serial ports and more. It has historically been the native method of file and print sharing for Microsoft operating systems, but has been replaced in modern OS:es by its successor protocol *Common Internet File System*, CIFS.

WINS: *Windows Internet Naming Service* provides a distributed database for registering and resolving dynamic NetBIOS names to IP address in a routed network environment. Without a WINS server, queries must be made via broadcast.

Appendix B

SM RemboAdmin User's Manual

B.1 Introduction

This chapter briefly introduces SM RemboAdmin and its main features followed by the prerequisites under which it will run, along with recommended reading and document notation.

B.1.1 Overview

SM RemboAdmin is a cloning wizard for written in Rembo-C for use with Rembo Toolkit 2.0 and later, a product by Rembo Technology SaRL (`http://www.rembo.com`). Its purpose is to benefit from the features of Rembo Toolkit in order to get a flexible cloning suite for Windows (NT/9x) and Linux, usable in large-scale environments. The difference between SM RemboAdmin and the default wizards supplied with Rembo Toolkit are threefold:

- SM RemboAdmin provides a graphical interface for creating and installing incremental images.
- SM RemboAdmin comes with support for grouping computers together by using options from DHCP. Each group have their own set of options that determine actions taken on boot and cloning related settings.
- SM RemboAdmin can be configured to perform a limited set of customization tasks for Windows NT clients.

This document describes installation, configuration and administration issues with SM RemboAdmin and related services. While SM RemboAdmin should work on any platform where Rembo Toolkit is supported, the required configuration during the installation will be exemplified from a Unix view.

B.1.2 Notation

This document uses the following notation to emphasize the meaning of words and phrases:

- Single words in the `Typewriter` face corresponds to a string of words handled by the computer, typically a filename, an URL or an Unix command.
- Words or sentences in *italics* are text shown in dialogues on the screen.
- Words in NOUN-STYLE denote proper nouns of software required for installation but not featured with SM RemboAdmin.
- Lines starting with the sign `>` are to be typed into a Unix shell, omitting the sign itself.

B.2 Installation

This chapter covers how to unarchive and modify the SM RemboAdmin source tree so that it can be compied and installed.

B.2.1 Prerequisites

Prior to continuing with the installation portion of this document, please make sure that your Rembo server is installed and fully functional. This part is described in the `INSTALL` document included in the Rembo Toolkit binary distribution. In particular, make sure the section "Server Setup" has been carried step by step and verify that the clients are booting before proceeding with this document.

SM RemboAdmin makes use of GNU MAKE and GNU TAR in order to unarchive and compile the source code during the installation. This software can be achieved from `http://www.gnu.org/software/make` and `http://www.gnu.org/software/tar`, respectively.

At the time of writing, SM RemboAdmin blindly assumes that the clients that should run Windows either run Windows 9x or Windows NT/2000/XP (if any Windows at all), but not two or more versions on different partitions. The Windows installation must be installed at the first primary partition on the first disk at all times. If Linux is used, it must always be located on the second primary partition on the first disk.

B.2.2 Additional sources of information

For more information on Rembo Toolkit and the installation procedure, please consider the Rembo Toolkit 2.0 Manual online at `http://www.rembo.com/rembo/docs2/rembo`. This documentation also contains a lot of information on different cloning concepts and how Rembo utilizes network resources. Furthermore, it gives detailed feature descriptions and contains the Rembo-C scripting reference manual. The support forum for Rembo Toolkit 2.0 (`http://www.rembo.com/support_forums.htm`) is also a good way of retrieving information on the product.

B.2.3 Unarchiving the source code

SM RemboAdmin is distributed in the Tar (Tape ARchiver) format, compressed with Gzip. Both commands `tar` and `gzip` are standard components of most Unix based systems. Change to a temporary directory of your choice and type the following in your shell:

```
> tar -xzvf /path/to/SM_RemboAdmin-2.x.tar.gz
```

If your system does not have the GNU version of the tar utility (e.g. Solaris), the command above may not work. Then try the following:

```
> gzcat /path/to/SM_RemboAdmin-2.x.tar.gz | tar -xvf -
```

B.2.4 Compilation and installation

In order to provide binary compatibility with later releases of Rembo Tookit, SM RemboAdmin is distributed as source code. However, compiling it is very easy. Start by changing directory to the one created when unarchiving the code in the previous section:

```
> cd SM_RemboAdmin-2.x
```

The only required step before compilation is to edit a single line in the file `Makefile` in this directory. This must be done in order to find `rbcc` (the Rembo-C compiler) as well as to determine the installation path of SM RemboAdmin. Assuming your copy of Rembo Toolkit is installed in `/usr/local/rembo-2.0`, change the `REMBOPATH` variable in `Makefile` to look like this:

```
REMBOPATH = /usr/local/rembo-2.0
```

After saving `Makefile`, make sure you have write access to the `REMBOPATH` tree, then compile and install with the following commands:

```
> make
> make install
```

If all went well, a message stating success will conclude the screen output. If not, verify your access privileges and that no default directories are missing in `REMBOPATH/files/global`.

B.3 Environment configuration

This chapter covers the configuration required when SM RemboAdmin is installed. We describe how to configure the DHCP and Rembo servers to make full use of group control and the administration interface.

B.3.1 DHCP configuration

For the SM RemboAdmin group control to work, a group identifier has to be transferred via DHCP to the Rembo client. For this purpose we have introduced a site-specific DHCP option called `option-135` which contains a string of text (the group identifier). Using ISC DHCP SERVER version 3, declare this option by adding the following to the global scope of the DHCP server configuration file (`dhcpd.conf`):

```
option option-135 code 135 = text;
```

Then, for each scope of computers, provide their value for `option-135`. In a system with static delivery of IP-addresses each host is a sub scope of their own and may look something like this in `dhcpd.conf`:

```
host taumaster {
   hardware ethernet 00:90:27:bd:8e:79;
   option option-135 "stafflinux";
   fixed-address taumaster.csee.ltu.se;
}
```

For more information on ISC DHCP Server and configuration for both current and previous versions, please navigate to `http://www.isc.org/sw/dhcp`.

B.3.2 SM RemboAdmin group control

The group identifier transferred via DHCP is used to locate a section of the SM RemboAdmin configuration file where the configuration parameters for this particular group is found. The file is called `SM_RemboAdmin.conf` and is located in the directory `REMBOPATH/files/global/scripts`, where `REMBOPATH` is the server installation directory provided in Section B.2.4. From the client it is located inside the directory with the Rembo URL `net://global/scripts`.

The file is comprised of sections, where each section begins with the group identifier enclosed in brackets on a single line, e.g. `[stafflinux]`. Each line following the section name is parsed as an option for the group, with the option name to the left and the value to the right of the equal sign (=). A hash (#) avoids parsing the rest of the line. Below is an example snippet of the file.

```
# Comment
[stafflinux]
  linux = true
[other]
  wintype = winnt
  patchsid = true
```

SM RemboAdmin also has a graphical interface for editing group settings. This is covered in Section B.4.15.

B.3.3 Rembo authentication

Another component requiring configuration to fully utilize SM RemboAdmin is the Rembo server itself. While still assuming the clients are booting properly,

the authentication options should be appropriately set in `rembo.conf` for each set of clients.

First of all, create a Unix group in the user database used by the Rembo server. We will make the members of this group able to log in to the SM RemboAdmin administration interface, i.e. they will be able to create and delete images on the server. so be careful. Assuming the name of your new group is `winadm`, put this snippet of text below the global configuration parameters in `rembo.conf`:

```
AuthLocalDomain remboadm {
   UserGroup "winadm"
}
```

Now, for each group of computers to be accessible by the group, add the `AuthDomain "remboadm"` to the group properties. The `Options admin` directive indicates that this group has the Rembo-C interactive prompt enabled, which is extremely powerful and thus a security concern. It is strongly recommended to create a separate `GROUP` with the prompt enabled containing only trusted MAC addresses. The value of `StartPage` is the default setting for Rembo Toolkit and should be left unchanged for all groups.

```
GROUP stafflinux {
   StartPage "cache://global/rembo.shtml"
   AuthDomain "remboadm"
   Options admin
   Host 00:90:27:bd:8e:79
   # More hosts follow here
}
```

Please note that the `GROUP` name `stafflinux` is not at all associated with DHCP `option-135`. However, keeping them the same helps out. In addition, note that the Rembo server can be configured for other authentication mechanisms than users of Unix groups (i.e. PAM). Read the "Authentication domains reference" in server administration part of the Rembo Toolkit 2.0 manual for details.

B.3.4 Domain authentication setup (optional)

SM RemboAdmin and its surrounding environment can, optionally[1], be set up to create trust accounts in an LDAP database for use with a Samba PDC. But since Rembo doesn't support communication over the LDAP protocol, remote execution of a supplied script creating this LDAP account must be triggered on demand of the client.

In order to be able to communicate with the outside world from the Rembo client, a TCP tunnel must be configured on the server to provide for the `TCPConnect()` and related Rembo-C functions. In the server configuration file `rembo.conf` we define a tunnel named `DomainDaemon`, which will redirect TCP connections from the client to port 7070 on the server (localhost), defined by the following text:

[1]This feature is not very flexible and might be unusable in many environments and should be treated only as an example of the customization capabilities of Rembo, although the solution is fully functional on the site which SM RemboAdmin was written for.

```
TCPTunnel DomainDaemon {
   RemoteHost "localhost"
   RemotePort 7070
}
```

In order to trigger our script we must set up `inetd` (Internet services daemon) to invoke it upon connection on the specified port. Start by putting the following line in the file `/etc/services`:

```
domaind     7070/tcp
```

With the service defined, append the following line in `/etc/inetd.conf`, where `/usr/local/rembo-2.0/domaind.pl` is the location of the script located in the root directory of the SM RemboAdmin tarball.

```
domaind stream tcp nowait root \
     /usr/local/rembo-2.0/domaind.pl domaind.pl
```

To avoid unauthorized creation of trust accounts, it is assumed by the script that a NIS map named `domaingroup` is set up, containing all fully qualified hostnames of trusted clients. In addition, it is recommended that a `tcpd` wrapper should be used with `inetd` to add connection restrictions on port 7070. However, configuration of these aspects goes beyond the scope of this document.

B.4 Administration interface

In this chapter introduces the graphical interface of SM RemboAdmin. Each menu option is described in detail with information on the relevant configuration options.

B.4.1 Overview

The interface is divided into two different menus. The boot menu in Figure 1 is shown by default, and the administrator menu in Figure 2 is shown given administrative credentials. On each screen a set of icons with related text are located, listing all available operations (modules). A single left click on the mouse initiates an operation, and a right click displays a help window for the particular operation, listing all relevant options and their settings in the current group. The boot menu acts as a normal boot loader, from which users can be granted access to the administrator menu by the authentication setup discussed in Section B.3.3. We refer to the menu items as modules in this text.

B.4.2 Boot Windows

This operation will boot your Windows partition, i.e. the first primary partition on the first hard drive. Before starting Windows the following actions will be taken:

- When `wintype` is set to `winnt`, the hostname provided by DHCP will be written to the registry as the new hostname and NetBIOS name.

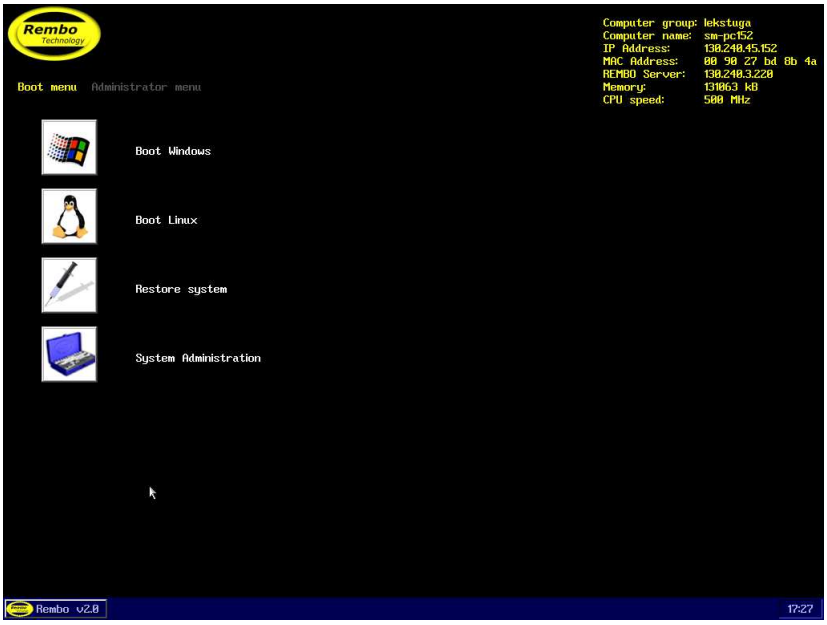

Figure 1: The boot menu.

- When `wintype` is set to `win9x`, the hostname will be written to the first line of he text file `C:\hostname`.

If the Windows partition doesn't exist, nothing will happen if you have option `preservepartitions = true`. Otherwise, your drive will be formatted and installed according to the relevant options for the "Auto-install machine" module on the Administration menu.

It is possible to run this operation automatically when the Rembo environment is loaded. Use option `autoboot = winnt|win9x` to achieve this.

This module is visible when `wintype = winnt|win9x`.

B.4.3 Boot Linux

This operation will boot your local Linux partition. Rembo will load the kernel and initial ramdisk specified by the options `defaultkernel` and `defaultramdisk`, respectively.

If the Linux partition doesn't exist, nothing will happen if you have option `preservepartitions = true`. Otherwise, your drive will be formatted according to the relevant options for the "Auto-install machine" module on the Administration menu.

It is possible to run this operation automatically when the Rembo environment is loaded. Use option `autoboot = linux` to achieve this.

This module is visible when `linux = true`.

B.4.4 Restore system

This operation is a special case of the "Auto-install machine" module in the Administrator menu, fully described in Section B.4.9. It is intended as a quick way of restoring an anonymous workstation to its original state, without formatting the partitions.

This module is visible when `showsystemrestore = true`. It should be visible only on machines without important local user data.

B.4.5 System administration

This operation will display the Administrator menu. Before the menu is loaded, Rembo will prompt for user name and password in order to verify it against the user database on the current Rembo server. The Administrator menu will vary a bit depending on the options `linux` and `wintype` of this group, determined by the visibility options associated with the administrator modules.

This module is always visible.

Figure 2: The administrator menu.

B.4.6 Auto-layout hard drive

This operation will display the hard drive partitioning/formatting dialogue. Information on the current structure of your first disk will be displayed, along with the possible future layout as described by option `partitions`. When confirming, the new partition table will be written and all partitions will be quick formatted. The Master Boot Record (MBR) is cleaned as well. **WARNING: Confirming this dialogue will erase all data on your disk!**

This module is always visible.

B.4.7 Partition editor

This operation will invoke the `FDisk()` call to Rembo, which starts the built-in tool for manual partition editing/formatting.

This module is visible if the current group has `Options admin` set in `rembo.conf`, as discussed in Section B.3.3. This limitation is Rembo Toolkit default and not easily changeable.

B.4.8 Manual image installation

This operation allows for manual selection of any combination of base images and any number of incremental images to be installed. After the images have been selected, the target drives will be formatted, without partitioning. During Windows NT/2000/XP installation (indicated by the `wintype` option), `patchsid = true|false` indicates whether the SID[2] should be patched or not.

This module is always visible.

B.4.9 Auto-install machine

This operation automatically reinstalls all configured software.

There are three ways this module can be invoked:

1. By clicking the "Auto-install machine" button in the Administrator menu
2. By clicking the "Restore system" button on the Main menu
3. By specifying the option `autoboot = reinstall` for this group

The module has two modes of operation:

- full mode - all involved partitions are formatted before applying disk images. Used in (1) above, and in (3) if `fullinstallmethod = true`.
- fix mode - disks are not formatted, yielding in quicker image restoration, but might fail in case of file system damage. Used in (2) above, and (3) if `fullinstallmethod = false`.

When partitioning and installing images (full mode), this module makes silent use of the hard drive layout module. Thus, the hard drive is initialized according to the `partitions` option. However, in order to protect the partition table, the option `preservepartitions = true|false` indicates whether to abort the operation in case the current partition table differs from the one specified by `partitions`.

When performing the actual image installation, this module makes silent use of the image installation module. At this point the `defaultlinbase` and `defaultwinbase` options determine the base images to be installed (depending on the operating systems selected with the `linux` and `wintype` options). During Windows NT/2000/XP installation, `patchsid = true|false` indicates whether the SID should be patched or not, and `applications` determine which incremental images to apply (if any) after base image installation.

This module is always visible.

[2]The SID (Security IDentifier) is a string of variable length from which user and group accounts are identified. By patching the SID we can make sure all computers keep an unique SID. Consider the Rembo Toolkit 2.0 Manual for a thorough description of this problem.

B.4.10 Netinstall Windows

This operation is used to load a large MS-DOS floppy disk image into RAM and boot it. Before this is done, Rembo will change the partition type of the Windows partition on the hard drive to FAT32 and format the drive, in order to be able to access it under MS-DOS. The RAM disk to be loaded is specified by the option `windowsramdisk`. The operation will not modify the partition table, only format the Windows partition.

The purpose with this operation is to provide a way to load a MS-DOS system with networking support and drivers for the known network interfaces on the site. When such a system is booted it is possible to start an unattended installation of e.g. Windows 2000 or XP over a SMB server share. **WARNING: After confirming this operation, all data on your Windows partition will be destroyed!**

This module is visible when `wintype = winnt|win9x`.

B.4.11 Create RAM disk from local drive

This operation will create a large bootable RAM disk to be used when installing Windows over a network (described in Section B.4.10).

This function removes the limited size of the conventional floppy by letting you pick a directory on local disk that will be imaged as your default boot disk. This way you can copy any floppy to disk and extend it with more files before creating this RAM disk.

However, to make a valid FAT floppy we need to merge the chosen directory with a blank, SYS:ed MS-DOS floppy specified by the Rembo URL `blankdosfloppy`, which should be located on the server. You will be prompted for insertion of such a disk during execution if it doesn't exist. The local directory and the blank floppy will then be merged to create the file `windowsramdisk`, with the old RAM disk renamed to extension ".old".

This module is visible when `wintype = winnt|win9x`.

B.4.12 Netinstall Linux

This operation will boot a Linux RAM disk with a specified kernel. The intention is to facilitate Kickstart installations of Red Hat Linux/Fedora.

If the Linux partition exists, the Linux ramdisk specified by the Rembo URL `linuxramdisk` will be loaded along with the server located kernel determined by the URL `installkernel`. The kernel will be loaded with the command `kernelcommand` which typically will contain the kernel argument `ks` with the the value being a Kickstart configuration on a NFS share, e.g. `ks=server:/path/to/kickstart.cfg`.

This module is visible when `linux = true`.

B.4.13 Create images

This operation will display a new window with a list of options for image creation.

- *Grab current Windows partition:* This option displays an input field for naming the Windows base image. The virtual memory files located in

their default locations will not be included in the image[3]. If the option `doublepartitions` is set, an image will be generated from the second primary partition on the first disk as well.

- *Grab current Linux partition:* This option displays an input field for naming the Linux base image. The contents of the temporary directory `/tmp` will not be included in the image.

- *Create application image with current disk:* This option creates a temporary base image of the current Windows partition and boots Windows. The next time SM RemboAdmin boots this particular client, the user will be prompted to supply a name for the image. Thereafter an incremental image reflecting the differences between the temporary image and the disk content is created. The temporary image is deleted afterwards.

- *Grab application image with base image:* This option creates an incremental image reflecting the differences between the current Windows partition and a base image selected via a file requester.

This module is always visible, but the available options may vary depending on the values of `linux` and `wintype`.

B.4.14 Delete images

This operation will display a new window containing a checkbox list of all disk images available to the current group of computers. This means all base images created from computers within this group, listed for the operating systems indicated by the `linux` and `wintype` options. In addition, incremental images for the current `wintype` created in any group will be visible. Deletion of any images are effective immediately and irreversible.

This module is always visible.

B.4.15 Configure SM RemboAdmin

This operation will launch a tool providing a graphical interface for the SM RemboAdmin configuration file introduced in Section B.3.2. Using this tool for configuration is recommended, not only because it is easier and more accessible as configuration can be performed from the client, but also because it helps avoid syntax errors in the configuration file. The entire tool is surrounded by a file lock that has to be acquired before configuration can start. If another client holds the lock, a warning will appear stating the user name, hostname, group and time of locking.

Figure 3 shows a snapshot of the interface. A pull-down menu is located at the top left of the window, where the user is able to change group to view and edit options for. The default group is the one this particular client belongs to, indicated by an asterisk (*). To the right of the pull-down menu buttons are located for deleting the selected group and adding another group.

[3]The intention is to save server storage. This kind of data is typically large files with different content on each machine and storing it has no purpose at all.

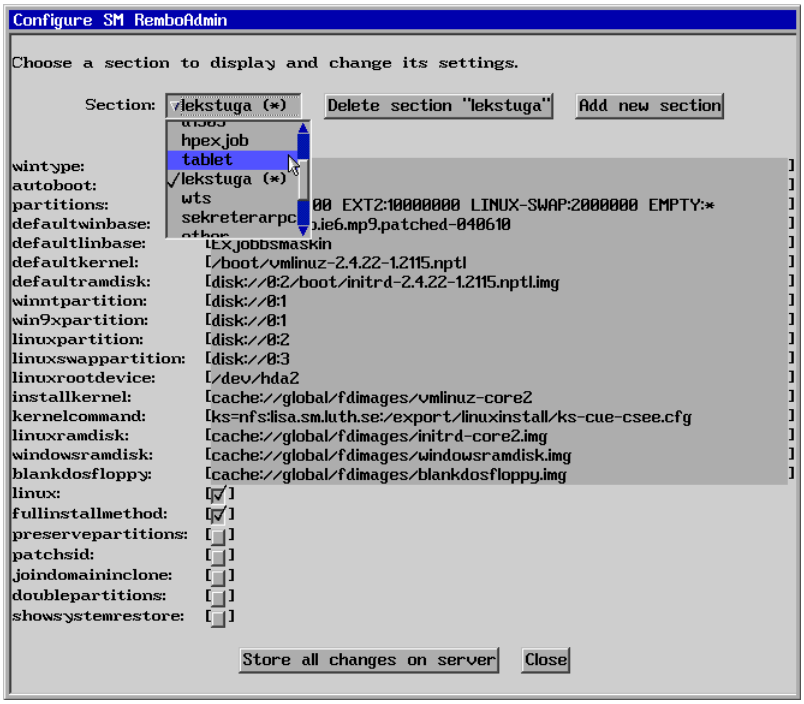

Figure 3: The SM RemboAdmin configuration tool.

The middle part of the window shows all[4] configuration options and their current values. The text fields are editable, and for the boolean options (where the value in `SM_RemboAdmin.conf` can be only `true` or `false`), checkboxes are present instead.

The bottommost part has two buttons; one labeled *Store all changes on server* and the second, *Close*, used to quit, ignoring any unsaved changes. The former will generate a new configuration file (with the old one backed up), followed a re-parse in order to verify and apply the new settings. Note that the new file will contain changes made to all sections. By clicking on *Close* the lock will be released, allowing other clients to configure.

This module is always visible.

B.4.16 Remove host-specific SID

This operation can be used to remove the host-specific SID entry on the server. When option `patchsid = true`, restoring Windows NT family images will generate a host specific file containing the unique SID to be applied to the Windows registry on all future restorations, in order to avoid having the SID in the base image on all machines. Before deleting the SID, a table will be shown with the SID in the Windows registry on disk, the SID in the default image (decided by

[4]The application option is not supported by this tool at the time of writing, i.e. there is no way to select incremental images. This will most likely be implemented by selecting from a checkboxed list (like the delete images operation) in the future.

the `defaultwinbase` option) along with the SID stored on the server.

Normally the same SID is kept over time, but when changing Windows versions on a computer (i.e. between Windows 2000 and XP) a new SID must be generated for compatibility reasons. Thus, we delete the SID on the server.

This module is visible if `wintype = winnt`.

B.5 Options summary

This chapter provides an overview of all options available for a computer group in SM RemboAdmin.

B.5.1 Table legend

In an attempt to make the the table smaller and less messy, the following symbols have been used when appropriate:

$: The value of this option must obey the partition tags described with the `GetPrimaryPartitions()` function in the Rembo-C Scripting Reference Manual.

-: The value of this option can be any string.

#: This option has no specified default value.

1: This value should be interpreted by prepending the string `cache://global/fdimages/` to it.

2: The presence of this option should be repeated for each desired value.

URL: The value of this option should be a file pointed to by a valid Rembo URL, i.e. a string beginning with `net://`, `cache://` or `disk://`.

Option name	Type	Valid values	Default value
wintype	string	winnt|win9x|none	winnt
autoboot	string	winnt|win9x|linux|reinstall|none	none
partitions	string	$	NTFS:10000000 EMPTY:*
defaultwinbase	string	-	Defaultwinbase
defaultlinbase	string	-	Defaultlinbase
defaultkernel	string	URL	disk://0:2/boot/vmlinuz
defaultramdisk	string	URL	#
linuxrootdevice	string	-	/dev/hda2
installkernel	string	URL	vmlinuz[1]
kernelcommand	string	-	#
linuxramdisk	string	URL	initrd.img[1]
windowsramdisk	string	URL	windowsramdisk.img[1]
blankdosfloppy	string	URL	blankdosfloppy.img[1]
linux	boolean	true|false	false
fullinstallmethod	boolean	true|false	true
preservepartitions	boolean	true|false	true
patchsid	boolean	true|false	true
joindomaininclone	boolean	true|false	true
doublepartitions	boolean	true|false	false
showsystemrestore	boolean	true|false	false
application[2]	string	-	#

Table B.1: SM RemboAdmin options summary.

Appendix C

SM RemboAdmin Maintenance Guide

C.1 Introduction

This chapter introduces this document and presents the file hierarchy upon which SM RemboAdmin is built.

C.1.1 Overview

SM RemboAdmin is a cloning wizard for written in Rembo-C for use with REMBO TOOLKIT 2.0 and later, a product by Rembo Technology SaRL (`http://www.rembo.com`). Its purpose is to benefit from the features of Rembo Toolkit in order to get a flexible cloning suite for Windows (NT/9x) and Linux usable in large-scale environments.

This document is intended as a guide to the SM RemboAdmin source code, enough for others to make modifications at their own need, and perhaps even develop it further. It will describe the main design and flow charts, without focusing on the Rembo-C[1] language in detail.

C.1.2 Notation

This document uses the following notation to emphasize the meaning of words and phrases:

- Single words in the `Typewriter` face corresponds to a string of words handled by the computer, typically a filename, an URL or an Unix command.
- Words or sentences in *italics* are text displayed in the graphical interface.
- Words in NOUN-STYLE denote proper nouns of other software.

[1] Rembo-C is fully detailed in the Rembo-C Scripting Reference Manual, available online from Rembo Technology SaRL.

C.2 Design overview

In this chapter we cover the main concepts of the design. The source code modules, the file hierarchy, naming conventions, flow charts and global structures will be presented.

C.2.1 Modules

The first thing to acquaint oneself with is the module concept of SM RemboAdmin. It is divided into about 15 different modules, despite the fact that dividing into modules has no semantic meaning in Rembo-C. However, it provides a way to separate GUI tasks into different source code files for a better overview of the code. Thus, the modular cohesion is mainly procedural, although many modules are more fine grained internally.

C.2.2 File hierarchy

With the modules in mind, let's have a look at the contents of the SM RemboAdmin tarball top level directory. A description of each file or directory is shown below.

```
INSTALL
Makefile
SM_RemboAdmin-help/
SM_RemboAdmin.conf
SM_RemboAdmin.rbc
SM_RemboAdmin.rbx
images/
settings.rbc
src/
```

INSTALL Instructions on how to install SM RemboAdmin. This procedure is more detailed in the SM RemboAdmin User's Manual.

Makefile File used by GNU MAKE to automate compilation and installation.

SM_RemboAdmin-help/ This directory contains a HTML file (a body) for each GUI task, i.e. roughly one file per module. These files are parsed and displayed by the special module `Help`. This directory will be installed in the `/scripts` directory on the server during installation.

SM_RemboAdmin.conf Configuration file example. The file will be installed in the `/scripts` directory on the server during installation if it isn't already present.

SM_RemboAdmin.rbc A file concatenated from all source modules. It is created during compilation (i.e. invocation of the `make` command), thus editing it will have no effect.

SM_RemboAdmin.rbx The executable file compiled from `SM_RemboAdmin.rbc`.

images/ Additional ZSoft Paintbrush (PCX) images used by the GUI, copied to the `/images` directory during installation.

settings.rbc A modified version of the same file provided with Rembo Toolkit, edited to run SM RemboAdmin instead of the default wizard. Installed in the `/scripts` directory during installation.

src/ This directory contains all `.rbc` files, one file of source code for each module.

C.2.3 Naming conventions

A module is always invoked by the function `ModuleName_main()`, located in its file `ModuleName.rbc`. The intention is to have globally accessible functions named on the form `Module_function()`, and otherwise use nested functions in Rembo-C to hide as many functions and variables as possible. However, many modules have GUI related functions exported globally, following the convention of having the module name prepended to the function name. Such functions are required to reside in the global scope, since the GUI response is invoked from different threads than the one calling on the module's main function.

C.2.4 Module flow diagram

The procedural cohesion of the modules has some limitations, and as SM RemboAdmin has grown the need for a more fine grained approach has emerged. This means that the code of certain modules is reused in different contexts by other modules. As an example, the hard drive may be partitioned using the HDInit module, but the module is also used during unattended installations via the Install module. Likewise, the Install module can be used both when manually selecting images and when performing unattended installations.

The modules used in more than one context behave differently depending on invocation. At this date there are four modules with dual functionality, namely ConfigParser, HDInit, Install and AutoInstall. Each takes a boolean argument to the main function in order to determine context. Figure 1 shows a simplified version of calls between modules and the triggering menu items. The most important branching conditions have also been included.

C.2.5 Global configuration structure

The Init module is, besides the fact that it initializes the entire program, a bit special in the sense that it contains the only[2] global data structure in SM RemboAdmin. This is the `struct MachineID { ... };` on top of the file `Init.rbc`. Each element of `MachineID` defines an option of type `bool` or `str` (string), where the name of the element is the same as its corresponding option name in `SM_RemboAdmin.conf`. The default values of all options are also defined with the structure definition.

The structure is declared as the variable `ThisHost` which is accessed directly throughout all modules, typically by branching on the options of type `bool`. Its

[2]In fact, the special option `application`, being a auto-growing array called `ThisHost_apparray`, could not be included in this structure since Rembo-C doesn't provide an equivalent to C-pointers.

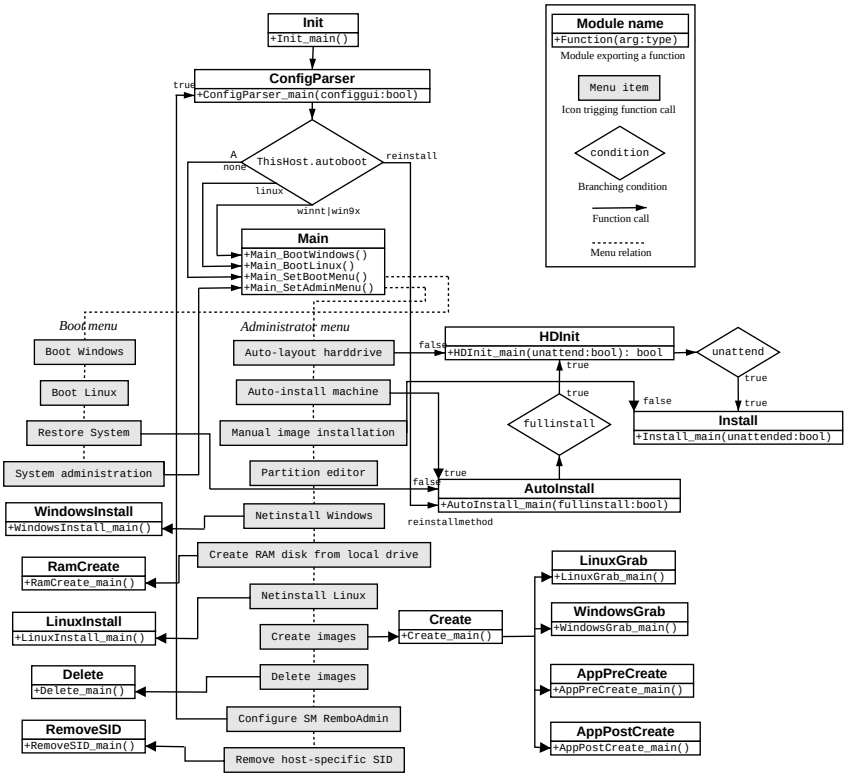

Figure 1: A flow diagram representing relations between menu items and modules.

name is a bit misleading, as the contents of `ThisHost` during execution will be the same for all computers in the group. The string elements (type `str`) are typically containers of file names. It is important to point out that the contents of `ThisHost` must not be modified anywhere in the code with the exception being the configuration tool, the *Configure SM RemboAdmin* administration option.

C.3 Further development

In this chapter some advice is given on extending SM RemboAdmin with options and menu items, followed by instructions on how to recompile and upgrade SM RemboAdmin. The chapter is concluded with ideas for improvements that could be implemented in the future.

C.3.1 Creating new options

Creating a new option for SM RemboAdmin is intended to be as easy as possible. The programming task is divided into three parts: Defining the option, declaring how it should be parsed and finally use it in the code to achieve the desired goal.

When modifications to the source code has been done, SM RemboAdmin has to be recompiled and upgraded as described in Section Section C.3.3.

C.3.1.1 Defining a new option

First of all, the option to be created must be a part of the `MachineID` structure in the Init module (`Init.rbc`). Assume a new boolean option called `registerhdd` is desired to indicate whether the hard disk drive should be registered after cloning or not. Put the following line of code inside the `MachineID {};` clause:

```
bool registerhdd = true;
```

This indicates that the value of `registerhdd` defaults to `true` for all groups unless something else is stated in the configuration. The default values should always be sane and set to reasonable values.

C.3.1.2 Configuration parsing

All options need to be parsed from the configuration file `SM_RemboAdmin.conf`, thus the second part requires modifications in `ConfigParser.rbc`. The function `ConfigParser_main(bool configgui)` contains a nested function `bool RegisterOption(str opt, str val)`, where all interpretation is done. Two arguments are required by `RegisterOption()`, the option name `opt` and the corresponding value `val`. It is called for each line for a specific group in the configuration file. Inside the `switch (opt) { ... }` clause we switch on `val` to interpret the string as boolean and insert it in `struct AllHosts`[3] as follows:

```
case "registerhdd":
   switch (val) {
   case "true":
      AllHosts.registerhdd = true;
      break;
   case "false":
      AllHosts.registerhdd = false;
      break;
   default:
   validvalues = "true | false";
   goto err;
}
break;
```

The string `validvalues` should contain all values accepted, as the `goto err;` statement will generate an informative error report based on it.

C.3.1.3 Extending the code

Third and finally, an example on how this could extend the code. The cloning and hard disk drive registration takes place in `Install.rbc`. A conditioning clause is put around the statements that perform the registration:

[3] `AllHosts` is a `MachineID` structure and stores what will be the `ThisHost` contents when the parsing is finished. It is used temporarily inside `RegisterOption()` as we need to parse more than one section at at time, using the very same code, when invoking the graphical configuration tool using `ConfigParser_main(true);`.

```
if (ThisHost.registerhdd) {
   with (HDDRegExceptionHandler) try {
      Win2KRegisterHDD(true);
      Win2KRegisterParts();
   }
}
```

When the new functionality is verified it is appropriate to add the new option to the help text in the file `Install_main.html` in the `SM_RemboAdmin-help` directory. Add the option both in the text and under the "Relevant settings" part of the file.

C.3.2 Adding a new module

The task of creating a module for SM RemboAdmin is divided into several steps, each one described below.

Start by adding an icon in any of the menus to be able to invoke the new module easily. This is done in `Main.rbc`. The boot and administration menus are defined in the `Main_SetBootMenu()` and `Main_SetAdminMenu()` functions respectively. Locate the code inside the appropriate function where the array `desktopmatrix` is modified. Insert a line on the following format in the desired position. The index of `desktopmatrix` indicate the order of appearance on the menu.

```
desktopmatrix[idx++] = {"NewModule_main();", "newmodule.pcx",
                        "A new module"};
```

The above suggests a new module defined by the function `NewModule_main()` using the icon `newmodule.pcx` with the accompanying text *A new module*. The icon's path is relative to the second argument of the `Main_SetDesktopContents()` function that follows the menu definition.

```
Main_SetDesktopContents("mainpanel", "cache://global/images",
                        desktopmatrix, 1);
```

Use the fourth argument to `Main_SetDesktopContents()` to adjust the number of columns to split the icons into.

In order to follow the convention of modules in Section C.2.4, create a new file in the `src` directory named `NewModule.rbc` containing a function defined as void `NewModule_main(void) { }`. To ensure this code is compiled with the rest of SM RemboAdmin, add the `NewModule` word to the `SUBMODULES` definition in the `Makefile`. The required recompilation and upgrade task that follows is described in Section C.3.3.

To provide help functionality for the new module, add a help file as described in Section C.3.1. Remember to document all configuration options used in the module!

C.3.3 Recompiling the source code

After desired changes have been made to the Rembo-C files in the `src/` directory, the code has to be recompiled. The new version of the Rembo executable `SM_RemboAdmin.rbx` has to be copied into the correct location in the Rembo Toolkit installation for the changes to be applied. On the top level of the SM RemboAdmin tarball, e.g. where `Makefile` is located, issue the following commands:

```
> make
> make updateexe
```

All clients launching Rembo after this operation will use the new version of SM RemboAdmin.

C.3.4 Possible improvements

At the time of writing there are some improvements that could be made in order to clean up and extend the code of SM RemboAdmin. To allow for more functional cohesion, a special utilities module should be created with functions that could be used in all modules. Some more examples:

- A general way to create checkbox lists and define their actions. Could be shared between the Delete and Install modules. This would easily provide a module that would just list all images available.
- Provide the possibility to have any operating system on any partition. Windows systems are currently required to be present on the first primary partition and Linux on the second due to the initial requirements on the application.
- Better exception handlers. It could be possible to configure whether certain critical exception handlers should send the administrator e-mail with error information or not.
- A way to configure general settings related to the application, not to a specific group. An idea is to add a `[global]` section to the configuration file and get rid of some constants in the code, e.g. the timeout before a client is automatically booted and the e-mail address to the administrator.

However, the key point with SM RemboAdmin is to provide a smooth cloning procedure. New deployment problems should lead the development.